1. (Currently Amended) An apparatus comprising:

at least one processor;

a memory coupled to the at least one processor;

a plurality of object oriented classes residing in the memory, at least one of the plurality of object oriented classes including state data that indicates a protected class;

a catalog of allowed classes residing in the memory, the catalog of allowed classes including all classes that are authorized to access at least one protected class; and

a state/domain checker residing in the memory and executed by the at least one processor, the state/domain checker performing a plurality of checks when each of the plurality of object oriented classes is loaded, the plurality of checks determining whether a class being loaded accesses the at least one protected class, and if so, determining whether the class being loaded is [authorized to access the at least one protected class] included in the catalog of allowed classes, and generating an exception if the class being loaded is not [authorized to access the at least one protected class] included in the catalog of allowed classes.

2. (Original) The apparatus of claim 1 wherein the state/domain checker further performs at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes.

3. (Currently Amended) [The apparatus of claim 2] An apparatus comprising:

at least one processor;

a memory coupled to the at least one processor;

a plurality of object oriented classes residing in the memory, at least one of the plurality of object oriented classes including state data that indicates a protected class;

a state/domain checker residing in the memory and executed by the at least one processor, the state/domain checker performing a plurality of checks when each of the plurality of object oriented classes is loaded, the plurality of checks determining whether a class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is authorized to access the at least one protected class, and generating an exception if the class being loaded is not authorized to access the at least one protected class, wherein the state/domain checker further performs at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes, wherein the at least one runtime check includes a check to determine whether a Java reflection method is invoked by a referencing class on a referenced class, and if a Java reflection method is invoked by the referencing class, and if the referenced class implements a private domain interface, and if the referencing class does not implement a system state interface, generating an error.

4. (Currently Amended) [The apparatus of claim 2] An apparatus comprising:

at least one processor;

a memory coupled to the at least one processor;

a plurality of object oriented classes residing in the memory, at least one of the plurality of object oriented classes including state data that indicates a protected class;

a state/domain checker residing in the memory and executed by the at least one processor, the state/domain checker performing a plurality of checks when each of the plurality of object oriented classes is loaded, the plurality of checks determining whether a class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is authorized to access the at least one protected class, and generating an exception if the class being loaded is not authorized to access the at least one protected class, wherein the state/domain checker further performs at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes, wherein the at least one runtime check includes a check to determine whether a Java Native Interface (JNI) function is invoked by an external program to access a protected class, and if the external program invokes a JNI function to access a protected class, and the external program is not running in system state, generating an error.

5. (Original) The apparatus of claim 1 wherein a class is a protected class if the class is defined as a private domain class or a system state class.

6. (Currently Amended) The apparatus of claim 1 wherein the plurality of checks includes a check during class verification that determines whether a class being verified implements a private domain interface or a system state interface, and if the class being verified implements a private domain interface or a system state interface, and if the class being verified is not included in [a] the catalog of allowed classes, generating an error.

7. (Currently Amended) The apparatus of claim [6] 1 wherein the catalog of [allowable] allowed classes is generated during a build process that packages the plurality of classes together into an installable form.

8. (Original) The apparatus of claim 1 wherein the plurality of checks includes a check during class preparation that determines whether a class being prepared has a superclass, and if the class being prepared has a superclass, and if the superclass implements a private domain interface or a system state interface, and if the class being prepared does not implement at least the same private domain interface or system state interface as the superclass, generating an error.

9. (Original) The apparatus of claim 1 wherein the plurality of checks includes a check during class resolution that determines whether a class being resolved to by a referencing class implements a private domain interface, and if the class being resolved to by the referencing class implements the private domain interface, and if the referencing class does not implement a system state interface, generating an error.

10. (Original) The apparatus of claim 9 wherein the check during class resolution is performed before runtime when a class is loaded.

11. (Original) The apparatus of claim 9 wherein the check during class resolution is performed at runtime when a method on the class being resolved to is invoked.

12. (Original) An apparatus comprising:

at least one processor;

a memory coupled to the at least one processor;

a plurality of Java classes residing in the memory, at least one of the plurality of Java classes including state data that indicates a protected class;

a Java Virtual Machine (JVM) residing in the memory and executed by the at least one processor;

a state/domain checker residing in the memory and executed by the at least one processor, the state/domain checker performing the following checks:

a first check during class verification that determines whether a class being verified implements a private domain interface or a system state interface, and if the class being verified implements a private domain interface or a system state interface, and if the class being verified is not included in a catalog of allowed classes that is generated during a JVM build process that packages the plurality of classes together into an installable form, throwing an exception;

a second check during class preparation that determines whether a class being prepared has a superclass, and if the class being prepared has a superclass, and if the superclass implements a private domain interface or a system state interface, and if the class being prepared does not implement at least the same private domain interface or system state interface as the superclass, throwing an exception;

a third check during class resolution that determines whether a class being resolved to by a referencing class implements a private domain interface, and if the class being resolved to by the referencing class implements the private domain interface, and if the referencing class does not implement a system state interface, throwing an exception;

(claim 12 continued)

a fourth check to determine whether a Java reflection method is invoked by a referencing class on a referenced class, and if a Java reflection method is invoked by the referencing class, and if the referenced class implements a private domain interface, and if the referencing class does not implement a system state interface, throwing an exception; and

a fifth check to determine whether a Java Native Interface (JNI) function is invoked by a program external to the JVM to access a protected class at runtime, and if the program invokes a JNI function to access a protected class, and the program is not running in system state, throwing an exception.

13. (Currently Amended) A method for creating and enforcing protected system level Java code comprising the steps of:

providing a catalog of allowed classes that includes all classes that are authorized to access at least one protected class;

loading a plurality of Java classes, each of the plurality of Java classes that is protected including state data that indicates a protected class; and

performing a plurality of checks when each of the plurality of Java classes is loaded, the plurality of checks determining whether the class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is [authorized to access the at least one protected class] included in the catalog of allowed classes, and generating an exception if the class being loaded is not [authorized to access the at least one protected class] included in the catalog of allowed classes.

14. (Original) The method of claim 13 further comprising the step of performing at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes.

15. (Currently Amended) [The method of claim 14] A method for creating and enforcing protected system level Java code comprising the steps of:

loading a plurality of Java classes, each of the plurality of Java classes that is protected including state data that indicates a protected class;

performing a plurality of checks when each of the plurality of Java classes is loaded, the plurality of checks determining whether the class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is authorized to access the at least one protected class, and generating an exception if the class being loaded is not authorized to access the at least one protected class,

performing at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes, wherein the at least one runtime check includes a check to determine whether a Java reflection method is invoked by a referencing class on a referenced class, and if a Java reflection method is invoked by the referencing class, and if the referenced class implements a private domain interface, and if the referencing class does not implement a system state interface, generating an error.

16. (Currently Amended) [The method of claim 14] A method for creating and enforcing protected system level Java code comprising the steps of:

loading a plurality of Java classes, each of the plurality of Java classes that is protected including state data that indicates a protected class;

performing a plurality of checks when each of the plurality of Java classes is loaded, the plurality of checks determining whether the class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is authorized to access the at least one protected class, and generating an exception if the class being loaded is not authorized to access the at least one protected class,

performing at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes, wherein the at least one runtime check includes a check to determine whether a Java Native Interface (JNI) function is invoked by an external program to access a protected class, and if the external program invokes a JNI function to access a protected class, and the program is not running in system state, generating an error.

17. (Original) The method of claim 13 wherein a class is a protected class if the class is defined as a private domain class or a system state class.

18. (Currently Amended) The method of claim 13 wherein the plurality of checks includes a check during class verification that determines whether a class being verified implements a private domain interface or a system state interface, and if the class being verified implements a private domain interface or a system state interface, and if the class being verified is not included in [a] the catalog of allowed classes, generating an error.

19. (Currently Amended) The method of claim [18] 13 wherein the catalog of [allowable] allowed classes is generated during a JVM build process that packages the plurality of classes together into an installable form.

20. (Original) The method of claim 13 wherein the plurality of checks includes a check during class preparation that determines whether a class being prepared has a superclass, and if the class being prepared has a superclass, and if the superclass implements a private domain interface or a system state interface, and if the class being prepared does not implement at least the same private domain interface or system state interface as the superclass, generating an error.

21. (Original) The method of claim 13 wherein the plurality of checks includes a check during class resolution that determines whether a class being resolved to by a referencing class implements a private domain interface, and if the class being resolved to by the referencing class implements the private domain interface, and if the referencing class does not implement a system state interface, generating an error.

22. (Original) The method of claim 21 wherein the check during class resolution is performed before runtime when a class is loaded by a Java Virtual Machine (JVM).

23. (Original) The apparatus of claim 21 wherein the check during class resolution is performed at runtime when a method on the class being resolved to is invoked.

24. (Original) A method for creating and enforcing protected system level Java code comprising the steps of:

running a Java Virtual Machine (JVM);

the JVM loading a plurality of Java classes, each of the plurality of Java classes that is protected including state data that indicates a protected class;

performing a first check during class verification that determines whether a class being verified implements a private domain interface or a system state interface, and if the class being verified implements a private domain interface or a system state interface, and if the class being verified is not included in a catalog of allowed classes that is generated during a JVM build process that packages the plurality of classes together into an installable form, throwing an exception;

performing a second check during class preparation that determines whether a class being prepared has a superclass, and if the class being prepared has a superclass, and if the superclass implements a private domain interface or a system state interface, and if the class being prepared does not implement at least the same private domain interface or system state interface as the superclass, throwing an exception;

performing a third check during class resolution that determines whether a class being resolved to by a referencing class implements a private domain interface, and if the class being resolved to by the referencing class implements the private domain interface, and if the referencing class does not implement a system state interface, throwing an exception;

performing a fourth check to determine whether a Java reflection method is invoked by a referencing class on a referenced class at runtime, and if a Java reflection method is invoked by the referencing class, and if the referenced class implements a private domain interface, and if the referencing class does not implement a system state interface, throwing an exception; and

(claim 24 continued)

performing a fifth check to determine whether a Java Native Interface (JNI) function is invoked by a program external to the JVM to access a protected class at runtime, and if the program invokes a JNI function to access a protected class, and the program is not running in system state, throwing an exception.

25. (Original) A method for building a computer program that includes system level code, the method comprising the steps of:

generating Java source code for a plurality of object oriented classes, each of the plurality of object oriented classes that is protected including state data defined in the Java source code that indicates a protected class;

identifying from the Java source code for each object oriented class which of the plurality of object oriented classes are protected;

compiling the Java source code for the plurality of object oriented classes, thereby creating a plurality of class files that correspond to the plurality of object oriented classes;

creating a catalog of allowable classes, the catalog including all protected classes;

compiling source code for a Java Virtual Machine (JVM) to produce an executable JVM;

creating installable media that includes the executable JVM, the catalog of allowable classes, and the class files.

26. (Currently Amended) A program product comprising:

a state/domain checker that performs a plurality of checks when each of a plurality of object oriented classes is loaded, the plurality of checks determining whether a class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is [authorized to access the at least one protected class] included in a catalog of allowed classes that includes all classes that are authorized to access at least one protected class, and generating an exception if the class being loaded is not [authorized to access the at least one protected class] included in the catalog of allowed classes; and

signal bearing media bearing the state/domain checker.

27. (Original) The program product of claim 26 wherein said signal bearing media comprises recordable media.

28. (Original) The program product of claim 26 wherein said signal bearing media comprises transmission media.

29. (Original) The program product of claim 26 wherein the state/domain checker further performs at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes.

30. (Currently Amended) [The program product of claim 29] A program product comprising:

a state/domain checker that performs a plurality of checks when each of a plurality of object oriented classes is loaded, the plurality of checks determining whether a class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is authorized to access the at least one protected class, and generating an exception if the class being loaded is not authorized to access the at least one protected class, wherein the state/domain checker further performs at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes, wherein the at least one runtime check includes a check to determine whether a Java reflection method is invoked by a referencing class on a referenced class, and if a Java reflection method is invoked by the referencing class, and if the referenced class implements a private domain interface, and if the referencing class does not implement a system state interface, generating an error.

31. (Currently Amended) [The program product of claim 29] A program product comprising:

a state/domain checker that performs a plurality of checks when each of a plurality of object oriented classes is loaded, the plurality of checks determining whether a class being loaded accesses at least one protected class, and if so, determining whether the class being loaded is authorized to access the at least one protected class, and generating an exception if the class being loaded is not authorized to access the at least one protected class, wherein the state/domain checker further performs at least one runtime check when a method that may reference a dynamically defined class is invoked and when a function is invoked that could potentially access a method on one or more of the plurality of classes, wherein the at least one runtime check includes a check to determine whether a Java Native Interface (JNI) function is invoked by an external program to access a protected class, and if the external program invokes a JNI function to access a protected class, and the external program is not running in system state, generating an error.

32. (Original) The program product of claim 26 wherein a class is a protected class if the class is defined as a private domain class or a system state class.

33. (Currently Amended) The program product of claim 26 wherein the plurality of checks includes a check during class verification that determines whether a class being verified implements a private domain interface or a system state interface, and if the class being verified implements a private domain interface or a system state interface, and if the class being verified is not included in [a] the catalog of allowed classes, generating an error.

34. (Currently Amended) The program product of claim [33] 26 wherein the catalog of [allowable] allowed classes is generated during a build process that packages the plurality of classes together into an installable form.

35. (Original) The program product of claim 26 wherein the plurality of checks includes a check during class preparation that determines whether a class being prepared has a superclass, and if the class being prepared has a superclass, and if the superclass implements a private domain interface or a system state interface, and if the class being prepared does not implement at least the same private domain interface or system state interface as the superclass, generating an error.

36. (Original) The program product of claim 26 wherein the plurality of checks includes a check during class resolution that determines whether a class being resolved to by a referencing class implements a private domain interface, and if the class being resolved to by the referencing class implements the private domain interface, and if the referencing class does not implement a system state interface, generating an error.

37. (Original) The program product of claim 36 wherein the check during class resolution is performed before runtime when a class is loaded.

38. (Original) The program product of claim 36 wherein the check during class resolution is performed at runtime when a method on the class being resolved to is invoked.

39. (Original) A program product comprising:

(A) a plurality of Java classes, at least one of the plurality of Java classes including state data that indicates a protected class;

(B) a Java Virtual Machine (JVM) executable program;

(C) a state/domain checker that performs the following checks:

(C1) a first check during class verification that determines whether a class being verified implements a private domain interface or a system state interface, and if the class being verified implements a private domain interface or a system state interface, and if the class being verified is not included in a catalog of allowed classes that is generated during a JVM build process that packages the plurality of classes together into an installable form, throwing an exception;

(C2) a second check during class preparation that determines whether a class being prepared has a superclass, and if the class being prepared has a superclass, and if the superclass implements a private domain interface or a system state interface, and if the class being prepared does not implement at least the same private domain interface or system state interface as the superclass, throwing an exception;

(C3) a third check during class resolution that determines whether a class being resolved to by a referencing class implements a private domain interface, and if the class being resolved to by the referencing class implements the private domain interface, and if the referencing class does not implement a system state interface, throwing an exception;

(C4) a fourth check to determine whether a Java reflection method is invoked by a referencing class on a referenced class at runtime, and if a Java reflection method is invoked by the referencing class, and if the referenced class implements a private domain interface, and if the referencing class does not implement a system state interface, throwing an exception;

(claim 39 continued)

(C5) a fifth check to determine whether a Java Native Interface (JNI) function is invoked by a program external to the JVM to access a protected class at runtime, and if the program invokes a JNI function to access a protected class, and the program is not running in system state, throwing an exception; and

(D) signal bearing media bearing the plurality of Java classes, the JVM executable program, and the state/domain checker.

40. (Original) The program product of claim 39 wherein said signal bearing media comprises recordable media.

41. (Original) The program product of claim 39 wherein said signal bearing media comprises transmission media.

## STATUS OF THE CLAIMS

Claims 1-41 were originally filed in this patent application. In response to the first office action dated 9/11/03, an amendment was filed on 12/10/03 that amended the specification, but did not amend any of the claims as originally filed. In the pending office action, claims 1, 5, 6, 8, 13, 17, 18, 20, 25-28, 32, 33 and 35 were rejected under 35 U.S.C. §103(a) as being unpatentable over the admitted prior art (APA) in view of U.S. Patent No. 6,381,734 to Golde. Claims 2, 14 and 29 were rejected under 35 U.S.C. §103(a) as being unpatentable over APA in view of Gold and further in view of U.S. Patent No. 6,397,384 to Briggs. Claims 9-11, 21-23, and 36-38 were rejected under 35 U.S.C. §103(a) as being unpatentable over APA in view of Gold and further in view of U.S. Patent No. 5,966,701 to Fresko. Claims 12, 14 and 39-41 were allowed. Claims 3, 4, 7, 15, 16, 19, 30, 31 and 34 were objected to as being dependent upon a rejected base claim, but would be allowable if properly rewritten in independent form. In this amendment, claims 1, 3, 4, 6, 7, 13, 15, 16, 18, 19, 26, 30, 32, 33 and 34 have been amended. Claims 1-41 are currently pending.